JitterNot: A Data-Driven MPC Based Scheme for ABR in VCAs

CS293N Early Project Report Writeup
Spring 2022

(Sections 1Abstract, 2Introduction, 3Related Work in Project Proposal [April 23 22] and final report)

4. Modeling data

In order to develop a data-driven model such as RNN or LSTM, a rich dataset is required that captures most of the network conditions. However, tackling a novel problem like ours (designing an ABR framework for VCAs) has the downside of no existing dataset that is publicly available. Thus, the dataset had to be curated independently with a provision for fine adjustment of the network link parameters to emulate different network conditions. Also, the neural network(s) we develop should be relatively uncomplicated for the model to fit the training data well.

MahiMahi's [1] network emulation tools can be used to emulate many different link conditions. Mahimahi supports emulating fixed propagation delays (DelayShell), fixed and variable link rates (LinkShell), and stochastic packet loss (LossShell). LinkShell also supports various queueing disciplines such as DropTail, DropHead, and active queue management schemes like CoDel. Each of Mahimahi's network emulation tools can be arbitrarily nested within one another, providing more experimental flexibility. Thus, MahiMahi was chosen as the tool for emulating various network conditions.

The webrtc-internals functionality has been used by WebRTC application developers to understand the features and functions of their WebRTC services. To the best of our knowledge, it is relatively new to explore and utilize these stats to study the Quality of Experience aspects of WebRTC services. The webrtc-internals functionality enables observation of the performance of the WebRTC connections locally in the browser. During a 10-day test period, we set up video conference sessions on Google Meet, a WebRTC application, in the Google Chrome browser running in the link, delay, and loss shells - running as nested instances to provision varying network conditions. A mix of network conditions was applied - latency from 0 to 1500ms, link packet loss from 0 to 35%, and link speed from 3 Mbps to 100mbps. Each video session was kept for 3-4 minutes giving the total volume of the video conference statistics dataset amounting to 6-7 hours.

The sampling resolution of the webRTC internals tool is 1 second, giving us over 25,000 data points each containing features giving various performance details of the video conference. Also, for each session, the same video was played in a video player offline in the test machines. This helped us to produce accurate results for two reasons: the network in the test machine had only the video conference traffic, and video segments with high movement scenes are likely to be encoded with a high bitrate before transmission, effectively having a variable bitrate. Having the same video played throughout the training data would ensure a good fit in our neural network models as each session will have a similar bitrate. Although this might not be ideal to generalize for other videos, we expect the resolution adjustment decisions performed by our model predictive controller to be more precise. This could be addressed by adding to the data set and using more real-world scenarios during a video conference.

5. System description

WebRTC's reference implementation has been incorporated into major Web browsers[3]. We conducted our sessions on Google Meet and used the Google Chrome browser. When using Google Chrome, data from both the sending and receiving parties in a WebRTC-based telemeeting can be gathered via the WebRTC internals page (chrome://webrtc-internals/). For our setup, we consider a two-party call wherein one peer (sender A) will "share screen" to stream a video being played on a local video player. While the receiver (peer B) is muted and also has his camera turned off.

Figure: A two-party video conference call opens four tracks

For the peer Connection between peers A and B, a unique SSRC ID (one per track) is shared between them. Peer A (sender), has stats labeled as sent (e.g.,bitsSentPerSecond) whereas peer B(receiver), has stats labeled as received (e.g., bitsReceivedPerSecond). We vary the network conditions for A, keeping the conditions for B unrestricted.

The RTCStatsReport for A will include a remote-outbound-rtp statistics object (of type RTCRemoteOutboundRtpStreamStats); it should also have a corresponding inbound-rtp object. Both of these provide information about the same batch of packets being transmitted from the remote peer to the local device. The difference is that remote-outbound-rtp describes statistics about the transmission(s) from the perspective of the remote peer (and so includes mostly empty or null values), while inbound-rtp offers statistics about the incoming data from the local peer's perspective [4]. Thus, the inbound-rtp provides the feedback of video quality as perceived at the receiver B.

The system in WebRTC or similar programs like Skype, and Hangout, generally include a video codec and transport protocol as independent subsystems, each with its own rate-control logic and control loop. The transport provides the codec with estimates of the network's data rate and the video encoder selects parameters (including a frame rate and bit rate) to match its average bitrate to the network's data rate. However, they suffer from a key limitation: they use fixed control rules or heuristics and don't take into account the deployment environment and network conditions.

We add our data-driven MPC control layer on top of the existing system that governs the outbound video stream resolution. And, sets the value for the same that optimizes the QoE formula (discussed later) after taking into account the throughput, fps, and jitter prediction - each modeled to provide accuracy in the deployment environment.

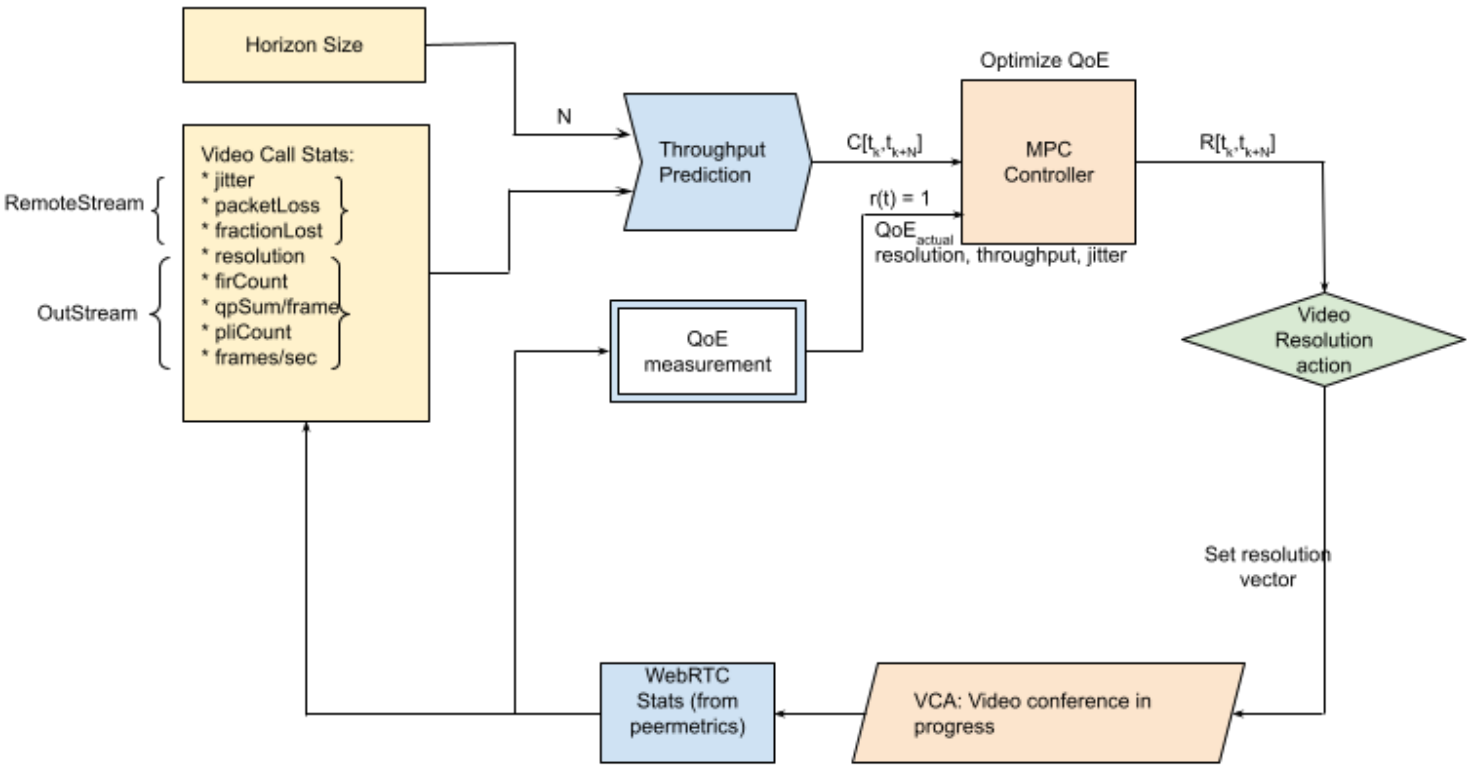

Figure: Block Diagram for JitterNot

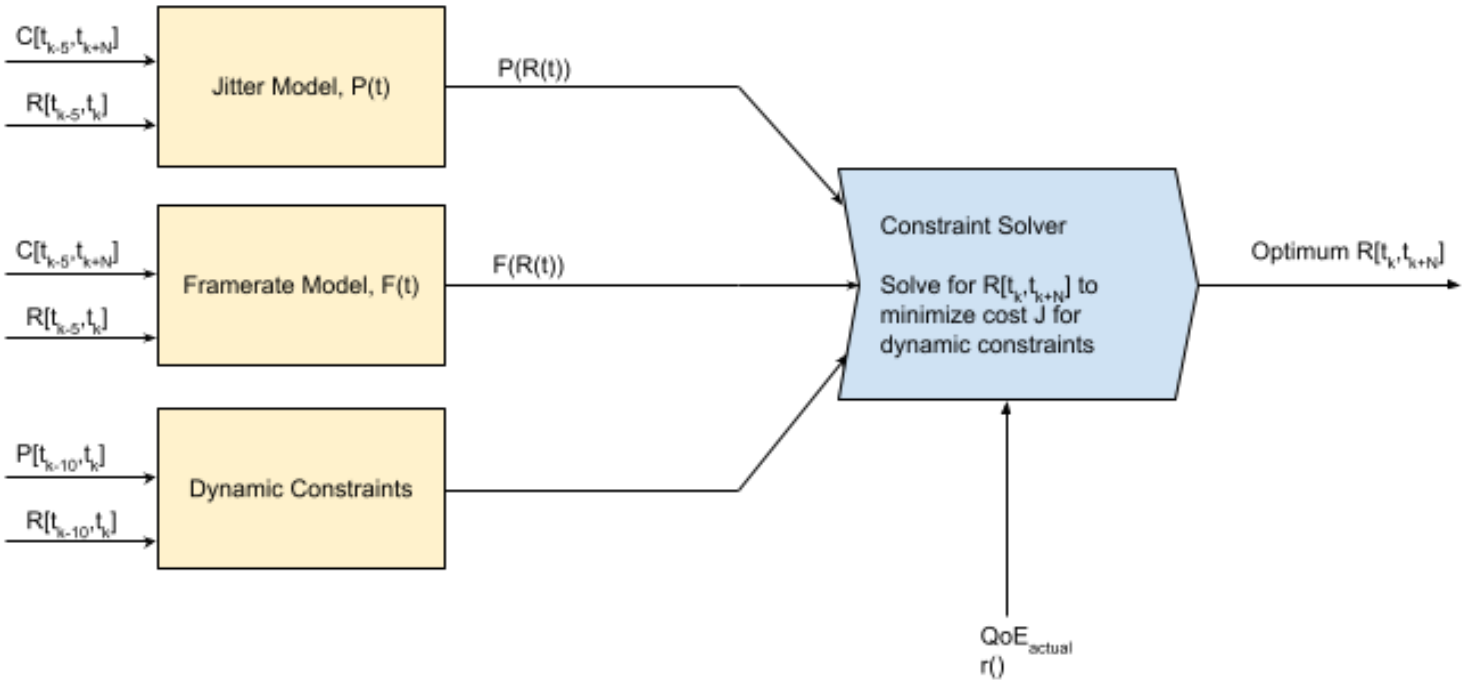

Figure: MPC Controller architecture

6. System Modeling

In this section, we design a model predictive control (MPC) method that can combine the throughput rate, fps, and jitter for resolution adaptation on the client that is streaming a video. We incorporate a Deep Learning model based on Recurrent Neural Network (RNN) for accurate throughput prediction and Convolutional Neural Network (CNN) models for jitter and fps estimation taking the predicted throughput into account.

6.1. Preprocessing of data** *[Under Development]*

6.2 LSTM model for throughput prediction** *[Under Development]*

6.3. Linear model for fps ** *[Under Development]*

Frame rate indicates the number of frames per second that are sent or received in a video stream. A typical video call will strive to work at 30 frames per second (fps), and that number will drop depending on the processing capabilities of the endpoints, bandwidth available, resolution of the video and network condition [6].

$$F(t) = c \times \frac{C(t)}{R(t)} + d \quad \text{, where c and d are model parameters}$$

6.4. Linear model for frame jitter ** *[Under Development]*

Jitter is the amount of variation in packet delay, which is why it is also frequently called delay variation. It is measured by calculating the average variation in packet arrival times at the receiving node occurring over a fixed interval. At callstats.io, the jitter calculation interval varies between five and ten seconds depending on endpoint configuration. [7]

Frame jitter can thus be derived from network jitter, by modifying the definition to measure delay variations for each frame instead of a single packet and so it varies with the resolution (or total pixels) of the frame R_k. We define R_k to be a product of frameWidth and frameHeight as it captures the amount of data or pixels in each frame.

$$P(t) = a \times \left(\sum_{k=t-5}^{t} \left| \frac{R(k)}{C(k)} - \frac{R(k-1)}{C(k-1)} \right| \right) - b, \text{ where a and b are model parameters}$$

6.5. Why MPC?

First of all, model predictive control may be naturally suitable for the problem of resolution or bitrate adaptation [5]. But we cannot say that MPC is the best choice among all possible ABR algorithms. We can only say that the MPC algorithm is more effective in mobile network video conferencing. Ideally, for a video call $[t_k, t_{k+1}]$, the QoE optimization problem can be directly calculated by the optimal resolution $R_1,\dots, R_k$ and jitter, and fps.

However, in practice, we do not have access to this perfect information, making it difficult to optimize the optimal solution offline. Despite the fact that perfect information for the future as a whole is not available, it is possible to obtain reasonably accurate throughput predictions over a short span of $[t_k, t_{k+N}]$ in the near future.

Owing to the network conditions being quite stable over a short period of time and not changing drastically within a few tens of seconds, which also holds for the video call metrics. So we can use this feature of throughput to run QoE optimization by applying the first resolution R_k and moving the horizon forward to $[t_k, t_{k+1}]$. This scheme is known as model predictive control (MPC). The advantage of MPC is that MPC can use prediction to optimize complex control objectives in dynamic systems online, subject to constraints.

6.6. Controller Architecture

The key idea of MPC is to maximize QoE (expressed in terms of video quality parameters) of users via a model predictive control method during the entire session. For each N second horizon, MPC first uses a DL-based throughput predictor to estimate future bandwidth $[C_k, C_{k+N}]$ for the horizon. It then calculates the jitter P_k and framerate F_k according to C_k from their respective NN models. These are expressed in terms of resolution R_k, which can obtain maximum QoE, a normalized multiplicative combination of these parameters, subject to dynamic constraints.

6.6.1 MPC Controller

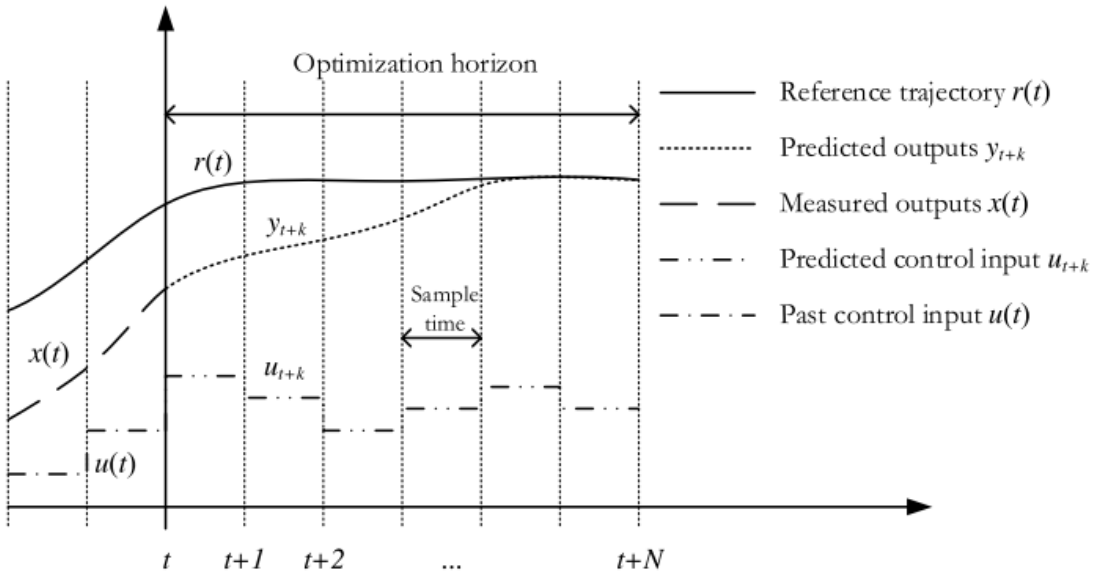

Fig. 2 shows inputs and outputs of the MPC controller [5]. MPC uses a system model to predict the future states of the system and generates a control vector (manipulated variable MV) that minimizes a certain cost function over the prediction horizon in the presence of constraints. The first element of the computed control vector at any sampling instant is applied to the system input, and the remainder is discarded. The entire process is repeated the next instant. Cost function can take the form of tracking error, control effort, jitter, low-resolution penalty, low framerate penalty, or a combination of these factors.

Constraints can be placed on the rate and range limits of manipulated variables (e.g., upper and lower limits of jitter, framerate, and range limits for resolution). This effort results in a controller that is robust to both time-varying disturbances and system parameters and regulates the process tightly within given bounds.

At each prediction horizon, the system output is also measured and is used as an initial condition in the optimization process. This helps in eliminating any un-modeled disturbances or modeling errors. The system model can be written in a variety of forms such as differential equations, state-space representation, transfer function representation, or discrete difference equations.

MPC performs the following three key steps: predict, optimize, and apply.
1) Predict: We use Deep RNN to predict throughput, simplify the jitter P_k and framerate F_k subsystems, and use respective CNN models for current values that only depend on the manipulated variable, resolution R_k.

2) Optimize: This is the core of the MPC algorithm: Given the previous resolution $R[t_{k-M},t_{k-1}]$, jitter $P[t_{k-M},t_{k-1}]$, framerate $F[t_{k-M},t_{k-1}]$, and throughput prediction $C[t_k,t_{k+N}]$, find optimal resolution R_k.

$R_k = f_{mpc}(R[t_{k-M},t_{k-1}], P[t_{k-M},t_{k-1}], F[t_{k-M},t_{k-1}], C[t_k,t_{k+N}])$, implemented by solving QoE_MAX. f_{mpc} also includes current jitter P_k and framerate F_k terms which are expressed using R_k as the input to their respective models.

3) Apply: Use the $R[t_k,t_{k+N}]$ control vector as resolution in the VCA for optimized QoE in the horizon $[t_k,t_{k+N}]$.

6.6.2 Problem Formulation

In this section, we formally introduce the problem of interest, give details of the constraints and cost function considered by our MPC controller, and how we design the adaptive resolution algorithm into the MPC framework.

Consider a multi-input single-output (MISO) plant S, with input u and output y signals sampled at a regular time interval Ts [6]. We aim at synthesizing a controller C for S such that the controlled system achieves a desired engineering objective defined in terms of minimization of a cost $J(y:T , \{u1:T\})$, where $u1:T$ denotes the sequence of input signals measured at time steps t = 1,..., T, and T is the length (measured in the number of samples) of the experiment where the performance is measured.

The inputs u1 to S in our problem will be jitter P, resolution R, and framerate F. However, as mentioned in section #, since we can model P and F in terms of R using CNN, our MISO system gets simplified to a SISO or single-input single-output system. The output y represents QoE is given by:

$$QoE = \left(\frac{P_{min}+0.5P_{avg}}{P(Rt)+0.5Pk_{avg}}\right)^2\left(\frac{Rt}{R_{max}}\right)\left(\frac{1+F(Rt)}{1+F_{max}}\right)^3$$

where avg, min, max are calculated from previous window $k\in[t\text{-}10,t]$

QoE is thus a normalized term and generally has a value in [0,1]. This is a product combination of jitter, resolution and framerate. The jitter term includes $P_k$min in the numerator, provides the controller more dynamic command over the plant. Similarly, for resolution and framerate terms. Framerate term has a constant 1 added to both the numerator and denominator so as to avoid a 0 by 0 occurrence.

Besides minimizing the cost $J(y:T , \{u1:T\})$, the following constraints on inputs and outputs should be satisfied:

$$umin \le u(t) \le umax \qquad (1)$$

which for our problem is given as:

$$0 \le P(t) \le 1.2Pmax, \qquad (1a)$$

$$Rmin/1.2 \le R(t) \le Rmax \qquad (1b)$$

$$1 \le F(t) \le 30 \qquad (1c)$$

Constraints (1) impose the domain limitation for the inputs and, gives little room for the performance metric to worsen since the optimization solution may exist wherein one or more inputs are worse but provides the optimal QoE under given condtions. The control design problem is formulated as the following optimization problem:

$$\min_{Rk\in R} \; J(y:T , u1:T) \qquad s.t.\ (1)\ , \qquad (2)$$

where the cost $J(y:T , \{u1:T\})$ is the summation of mean squared error (MSE) over the horizon for r(t) = 1 as the reference trajectory, since value 1 for the normalized QoE is desired.

$$J(y:T , u1:T) = \frac{1}{N}\sum_{k=t}^{t+N}(y(k) - r(k))^2$$